

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

## **IMAGES ARE BEST AVAILABLE COPY.**

As rescanning documents *will not* correct images,  
Please do not report the images to the  
Image Problem Mailbox.

**THIS PAGE BLANK (USPTO)**

**PRIORITY  
DOCUMENT**  
SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH RULE 17.1(a) OR (b)



REC'D	15 MAR 1999
WIPO	PCT

**Bescheinigung**

EP 98/08507

Die ACOS International Limited Central Office in Dublin/Irland hat eine Patentanmeldung unter der Bezeichnung

"Outside-Integration von Softwarekomponenten"

am 2. Januar 1998 beim Deutschen Patent- und Markenamt eingereicht.

Das angeheftete Stück ist eine richtige und genaue Wiedergabe der ursprünglichen Unterlage dieser Patentanmeldung.

Die Anmeldung hat im Deutschen Patent- und Markenamt vorläufig das Symbol G 06 F 17/00 der Internationalen Patentklassifikation erhalten.

München, den 4. Februar 1999

**Deutsches Patent- und Markenamt**

**Der Präsident**

Im Auftrag

Aktenzeichen: 198 00 102.9

Hoß



2

## OUTSIDE-INTEGRATION VON SOFTWAREKOMPONENTEN

### Ausgangssituation / Zielsetzung

L  
Darf nicht geändert werden

Die Firma ROX Lufttechnische Gerätebau stellt auftragsbezogen Klimageräte her, deren Auslegung, Konfiguration und Produktion eine sehr aufwendige Gerätekonstruktion und Variantenauslegung erfordert. Da ROX entgegen einer Modularweise mit festen Längen der Funktionsblöcke die Gerätegrößen den kundenindividuellen Wünschen anpassen kann, war ein sehr hoher Konstruktionsaufwand für die Konstruktion der Klimagerätegehäuse erforderlich.

Ziel war die automatisierte Konstruktion von Gerätegehäusen, die Variantenauslegung, die Erstellung aller Fertigungsunterlagen incl. technischer Zeichnungen sowie die direkte Ansteuerung einer Laser-Blechschnideanlage in einem geschlossenen Kreislauf zu realisieren (CIM-Konzeption). Die maximal mögliche Nutzung von Rationalisierungspotential sollte erreicht werden, indem das System expertensystemorientiert arbeitet und u.a. automatisch DXF-Dateien generiert, die dann für den Zeichnungsausdruck in gängigen CAD-Systemen importiert werden können. Dies ist konträr zur üblichen Vorgehensweise, nämlich über CAD und Variantenstücklisten konstruktiv die Fertigungsunterlagen zu bestimmen. Die Verbesserung der existenten Vorgehensweise über CAD und Variantenstücklisten versprach keinen nennenswerten Rationalisierungserfolg.

Es war demnach unabdingbar,

- a) daß Wissen der Konstrukteure in einer Wissensbasis, die aus einer Vielzahl von Tabellen, Regeln und Consultationskomponenten besteht, zu archivieren. Ziel war 80 Prozent des Produktionsvolumens abzudecken. Die Einrichtung der Wissensbasis für einmalig vorkommende Sondergeräte gibt keinen Sinn. Die Wissensbasis muß kognitiv (lernend) im Rahmen einer Consultationssitzung interpretiert werden, damit der Techniker schnell und sicher durch die checklistenorientierte Befragung zur Generierung aller auftragsbezogenen Fertigungsunterlagen geführt wird.
- b) Verfahren zu entwickeln, um Varianten und deren Ausprägungen (z.B. Maße, Gewichte, Preise,..) auf einfachste Art durch Techniker via Editor einrichten zu können. Da infolge technischer Änderungen permanent mit Erweiterungen/Änderungen von Ausprägungen ausgegangen werden muß, sind datenmodellabgeleitete Verfahren - die im Ergebnis Stammdatenbeschreibungen liefern - nicht geeignet. Neue zu berücksichtigende Varianten und deren Ausprägungen erfordern neue Attribute, die sich bei herkömmlicher Verarbeitung auf Stammdaten und Stammdatenverwaltungsprogramme auswirken und damit einen ständigen Programmwartungsaufwand erfordern. Die Einrichtung und Pflege von tausenden Stammsätzen ist auf Dauer zu kostenintensiv.

Um die gesamte Problematik handhabbar zu gestalten, wurde eine Systemmodellierung entsprechend Abb.1 gewählt. Abb.1 zeigt die realisierte Systemaufteilung sowie Interaktionen und Referenzierungen zwischen Objekten.

Objekte sind technisch oder betriebswirtschaftlich orientierte vorgefertigte Komponenten wie z.B. die Komponente Ventilatorauslegung, Taschenfilterauslegung, Wärmetauscher auslegung, Auskunft ffd. Aufträge, Verfügbarkeitsprüfung,... Damit wird der Objektbegriff - im Gegensatz zu objektsprachenorientierten Objekten in der z.T. kleinste Programmeinheiten wie die Aufteilung einer arithmetischen Operation in Objekte unterteilt werden (vgl. SMALLTALK) - edv-unabhängig und ausschliesslich anwendungsorientiert verwendet. Mit solchen Komponenten werden je nach Anwendungsbedarf Geschäftsprozesse wie z.B. die Gerätekonfiguration, die Wareneingangsbearbeitung, der finanzbuchhalterische Monatsabschluß, .. konfiguriert. Die Komponenten werden „Metasystemunterstützt“ automatisch nach objektorientierten Softwareproduktionsmethoden produziert. In Abb.2 wird schematisch der Aufbau einer Komponente dargestellt.

Der Wandel vom Objekt zur Komponente bedeutet, den Fokus eher auf eine Aggregation aller involvierten Komponenten, als auf die Spezialisierung der einzelnen Komponenten zu legen. Aggregation erfordert gegenüber der Spezialisierung zusätzliche softwaretechnische Grundlagen.



Bei Erstellung der vorliegenden Ausarbeitung mußte aus Platzgründen entschieden werden, ob die Diskussion von Grundlagen und Vorgehensweise der expertensystemunterstützten Konstruktion und Variantenauslegung Schwerpunkt ist, oder ob softwaretechnische Grundlagen der verwendeten Tools beleuchtet werden sollen. Da in Bezug auf Softwareproduktionsmethoden Grundlagenentwicklungen mit im Ergebnis neuen Möglichkeiten für die Softwareproduktion entstanden sind, wird in diesem Beitrag die Darstellung der Softwareproduktion priorisiert. Aus diesem Grunde wird das erzielte Ergebnis vor der Diskussion softwaretechnischer Grundlagen vorweggenommen:

## Ergebnis

Die durchschnittliche erforderliche manuelle konstruktions- und auslegungsbezogene Bearbeitungszeit für ein Gerät wurde von mehr als einem Tag auf weniger als eine Stunde reduziert! Die Projektkosten (1.200 TDM) waren innerhalb eines Jahres amortisiert. Das System ist seit 1988 im Praxiseinsatz und wird ausschliesslich von Technikern softwareseitig an den jeweils klimatechnisch bedingten Entwicklungsstand der Geräte angepaßt. Die Softwareumstellung auf eine technisch neue Klimageräteserie konnte 1996 innerhalb von 6 Wochen erfolgen. Es wird mehr als 80 % des Produktionsvolumens über das System bearbeitet. Aufgrund des Erfolges wurde die Software für Prime, IBM9370, AS400, RISC6000, MS-DOS, MS-WINDOWS, WINDOWS95, WINDOWS NT migriert. Seit 1988 wird das zugrundeliegende Metasystem in Bezug auf objekt- und komponentenorientierte Softwareproduktion optimiert.

## Wege entstehen durch Laufen

Die Transformierung von technischem Know-How auf Programmierer heute in der Vergangenheit dazu geführt, daß umfangreiche Software ab einem gewissen Änderungsaufwand in einen unwartbaren Zustand abglitt. Diese Situation konnte auch durch den Einsatz von Informatikern nicht geändert werden.

Grundvoraussetzung der neuen Softwaregeneration war, daß die Software komplett direkt von Technikern und nicht Programmierern erstellt und gewartet wird !!

Die Realisierung dieser Zielvorstellung versprach neben einer drastischen Kostenreduzierung (Einsparung von EDV-Personalkosten) die Option darauf, technisch bedingte Änderungen direkt in der Software abbilden zu können.

Die Analyse vorhandener Tools und Softwarearchitekturen führte zu folgenden ermittelnden Erkenntnissen

- a) Objektsprachenorientierte Softwareproduktionsverfahren genügen den Anforderungen nicht, da
  1. Basis für Vererbung und Polymorphismus ein statischer Vererbungsgraph ist
  2. Aggregationen unterstützende Sprachkonstrukte nicht in dem geforderten Umfang zur Verfügung stehenVererbungsgraphen müssen durch sehr zeitaufwendige Konstruktion einer Objektklassenstruktur bereits in der Designphase definiert werden. Sobald in diese Struktur neue Knoten eingefügt werden, kippt u.U. das Softwaresystem. Namhafte Fachautoren raten aus diesem Grunde davon ab, Vererbung in z.B. C++ zu verwenden (vgl. Nicolai Josuttis, OBJEKTspektrum Ausgabe Mai/Juni 1996, Die Wahrheit über Vererbung in C++). Gerade in der Variantenbestimmung ist bei technischen Änderungen davon auszugehen, daß Strukturänderungen erforderlich werden. Erschwerend kommt hinzu, daß reihenfolgeabhängige Sachverhalte (z.B. die Varianten von Funktionsblock X1 sind unterschiedlich je nachdem Funktionsblock X2 in der konkret auszulegenden Klimaanlage vor oder nach Funktionsblock X1 montiert wird) in einer statischen Struktur nicht abbildbar sind.
- b) Die Komplexität objektsprachenorientierter Softwareproduktionsverfahren ist einem Techniker nicht zuzumuten. Die Grundvoraussetzung war daher mit herkömmlichen Methoden nicht erreichbar. In einer 1996 von „Forrester Research“ in den USA durchgeführten Studie (Forrester: „Populistische“ Komponenten lösen elitäre Objekte ab) wird prognostiziert, daß die einen hohen „Skill“ voraussetzenden objektorientierten Sprachen ab 1999 durch untereinander vertragliche Komponentenmodelle ersetzt werden. Zitat: „Die Objekt-Gurus werden zumindest beim Übergang von der elitären zur „populistischen“ Komponentenindustrie benötigt“.

Die Vorteile objektorientierter Softwareproduktion gegenüber prozeduraler Softwareproduktion sind unbestritten, zwingen dennoch nicht zur Verwendung einer bestimmten objektorientierten Sprache oder Methode. Gerade die aktuelle Sprachen- und Methodendiskussionen (Eifel vs. C++, C vs. C++, C++ vs. Smalltalk, Java vs. C++, VisualBasic vs. Delphi, JavaBeans vs. ActiveX) zeigen die Brisanz auf, sich von der gerade modernsten Sprache oder Methode – und den damit verbundenen Interessen von sich wie Könige bekämpfenden Gruppierungen – abhängig zu machen. Für Unternehmen ist es nahezu unmöglich, in einem derart instabilen Umfeld zu planen.

Die ständige Neuentwicklung ein und derselben Applikation ist pure Narrerei. Die fuer Leser anmutende atypische Entscheidung COBOL als Sprache fuer die Entwicklung des im folgenden erwachten Metasystems zu verwenden, hat sich im nachhinein - insbesondere wegen der Risikominimierung durch Nutzung der weltweit verbreiteten Sprachgrundlage (nahezu 80 % der Business-Programme sind in COBOL geschrieben) - als richtig erwiesen. Waehrend der Entwicklungszeit sind einige hochgelobte Sprachen zunehmend bedeutungslos geworden. Der verwendete Compiler garantiert einerseits eine binaerkompatible Verfuegbarkeit des Metasystems auf ueber 700 Plattformen (write once run anywhere) und wird weltweit von ueber 1.000.000 Endbenutzern mit umfangreichen Business-Applikationen benutzt, andererseits besteht wegen der Verlagerung der Abhaengigkeit auf Compilerlieferanten praktisch keine Abhaengigkeit zum Toollieferanten. Natuerlich mussten die nicht in der Sprache existenten objekt- und komponentenorientierten Konstrukte mit einem hohen Entwicklungsaufwand - der im Ergebnis die Nutzung objekt- und komponentenbasierte Softwaretechnologien in einer separaten Schicht sprachen- und methodenunabhaengig ermöglicht - erkaufte werden. Dieser Aufwand hat sich jedoch durch problemlos und praktisch kostenlos durchgefuehrten Portierungen, sowie extrem kurze Realisierungszeiten neuer Projekte, mehrfach amortisiert.

Fundamentale Aenderungen in der Softwareproduktion waren erforderlich und erzwangen die Entwicklung neuer oder die Umkehrung vorhandener Denkansetze. „Wege entstehen durch Laufen“. Ein philosophischer Spaziergang: „Fallen wurde durch Fliegen ersetzt, Entfernungen werden durch Netze ersetzt, intelligente Facharbeit wird durch dumme Roboter ersetzt, Taylorismus wird durch Lean Production ersetzt, von Logopartnern zu parametrierende Molochs werden durch aus Leistungsbanken konfigurierbare Anwendersysteme ersetzt,... Die Ergebnisse dieser kybernetischen „Zeitprozesse“ - die Kulturen entstehen und verschwinden lassen - zu dokumentieren, bleibt Historikern ueberlassen. Im vorliegenden Anwendungsfall fuehrten Ersetzungen von herkömmlichen Denkansetzten und Softwareproduktionsmethoden dazu, daß direkt von Mitarbeitern der Fachabteilung umfangreiche und komplexe Software erstellt wurde und wird.

Basis hierzu ist die entwickelte vorwaertsgerichtete Expertensystemumgebung ALEXIS (Auslegungs-

expertise und Informationssystemumgebung). Grundlage fuer die Einrichtung von ALEXIS ist, daß eine vorhandene Erzeugnisstruktur direkt in einer Komponentenstruktur abgebildet werden kann. Es wird auf je Art einer statisch zu definierenden Programmablaufstruktur verzichtet, womit die mit der Synchronisation unterschiedlicher Strukturen (naemlich Erzeugnisstruktur und Programmablaufstruktur) verbundene Problematik - die Ursache fuer das Scheitern vieler Softwareprojekte war und ist - eliminiert wird. Der konkrete Programmablauf (Consultationsablauf) ergibt sich dynamisch, indem zu Beginn der Consultationsitzung die Bedienungskraft die technischen Funktionsbloেকে - und deren Montagereihenfolge in dem auftragsbezogenen Klimagesaet - ALEXIS mitteilt.

Als Traegersystem fuer ALEXIS wurde das Metasystem OBJECTline entwickelt. Dieses Metasystem kann quasi als API (Applikation Interface) in Komponentenskripten benutzt werden, indem auf die vom Metasystem durchzufuehrende Leistung in Komponentenskripten referenziert wird (Constrained Programming). Aus Sicht des Entwicklers von Softwarekomponenten wird im Gegensatz zu via Programmierung bedienender Schnittstellen von Objektklassen so weit wie moeglich auf erforderliche Schnittstellenprogrammierung verzichtet. Dies wird u.a. erreicht indem Schnittstellen in Aufgabentraegern (Tabellen, Bildschirmformaten...) gekapselt werden (z.B. wird in einer Tabelle neben den Daten fuer Varianten... die Schnittstelle fuer den Zugriff auf die Tabellendaten durch einen Tabellensteuersatz in der Tabelle selbst parametriert, oder einem Eingabefeld einer Bildschirmmaske wird programmextern eine Schnittstelle zur feldbezogenen Bearbeitung des Feldinhaltes injiziert). Neben der Containerfunktion sind Schnittstellen zur Realisierung des Datenbesorgerprinzips mit eigener - verwendungsbezogen ueberlagerbarer - Logik konfigurierbar. Dies ist Voraussetzung dafuer, daß Komponentenmonteure - unabhaengig von Komponentenlieferanten - Datenbe- und Datenenisorgungen in den zu adaptierenden Komponenten umlenken koennen. Die Unterscheidung zwischen Implementierung und Schnittstellen bewirkt, daß bei Aenderungen/Erweiterungen von z.B. Tabellen oder der Injizierung einer neuen Methode fuer z.B. eine Feldbearbeitung weder die Tabellenbearbeitung noch die Bildschirmformate aktivierenden Programmskripte (sowie die Bildschirmformate) manuell geaendert werden muessen.

Die auf einem Metasystem (ein Metasystem steuert seine eigenen Systembestandteile weitgehend kognitiv eigenstaendig) basierende Softwareerstellung benötigt immer weniger edv-bezogenes Spezialwissen, da das Metasystem generell erforderliche edv-bezogene Operationen wie Bildschirmsteuerung, Drucksteuerung,



## Plug\_in-Programming

Die Outside-Integration erfordert einen neuen Denkansatz, der zu den folgend dargestellten Grundlagen führt. Motivation zur Umsetzung des Denkansatzes war die intensive Beobachtung der Natur. In der Natur entstehen Organismen durch Injizierung und Instanziierung von materialisiertem Wissen in interpretierende Container. Dies gilt fuer organische Wesen (Menschen, Tiere, Pflanzen). Plug\_in-Programming ist eine - wenn auch sehr eingeschränkte - Nachahmung dieses Szenarios.

**Plug\_in-Programming** ermöglicht die programmexterne Konfiguration von Frameworks mit vorgefertigten Komponenten durch

1. Automatische Injizierung der MESSAGE's für neu hinzuzufügende Komponenten in vorhandene Systeme über Vererbungsparameter
2. Instanziierung von Komponenten in vorhandene Systeme (existente Workunits) durch Synchronisation mit dem vorhandenen System mittels Dynamic-Storage-Linking
3. Umkehrung von Datenversorgerprinzip in Datenbesorgerprinzip

'Programmextern' bedeutet, daß in vorhandene Softwaresysteme *ohne manuellen Programmieraufwand* - Komponenten injiziert und instanziiert werden. Hierbei wird a priori davon ausgegangen, daß Sourcen (Komponentenproduzenten erwarten den Schutz von Programm Sourcen) - fuer die Aggregation von Komponenten zu Geschäftsprozessen - nicht ausgeliefert werden. Der Komponentenmonteur kann unabhängig vom Komponentenanlieferanten Interface fuer die Verbindung der Komponenten spezifizieren und injizieren.

Folgende neue Grundlagen kommen bei Einsatz von Plug\_in-Programming zur Anwendung:

### Umkehrung von reagierender Programmierung in agierende Programmierung

Die neu zu instanziiierende Komponente injiziert die MESSAGE für seinen Ablauf selbst und ist damit agierend ! Diese Betrachtung transformiert reagierende statische Systeme in einen dynamisch handelbaren Zustand. Ergebnis im Anwendungsfall: Neu erforderliche Tabellen fuer die Auswahl von Variantenausprägungen werden via Editor aufgebaut, mit einem Vererbungsparameter versehen und in die vorhandene Software injiziert. Bei der danach folgenden Consultationssitzung wird der Bedienungskraft via Pull-Down-Menue die Tabelle zwecks Auswahl einer auftragsbezogenen Variantenausprägung angeboten

In der reagierenden Programmierung wird eine Komponente von einer anderen durch manuelle Codierung der entsprechenden MESSAGE *komponentenname.methodenname* aktiviert. Softwaremodifizierungen erfordern daher zwingend Änderungen in der vorhandenen Software. Es ist daher bei Erweiterungen eines Softwaresystems intensive Kenntnis ueber die vorhandenen Software erforderlich.

Injizierung: Die Message, die den Ablauf der Komponente veranlaßt, wird eingetragen  
Instanziierung: Die in Skriptsprache definierte Komponente wird in eine Runtimekomponente übersetzt und in eine Zielumgebung (Workunit) instanziiert (eingebunden). Injizierungs- und Instanzierungszeitpunkt sind zeitgleich.  
Dieselbe Runtimekomponente läuft unverändert auf allen Plattformen, auf denen auch das Metasystem abläuft.

### Umkehrung von Versorgerprinzip in Besorgerprinzip

Komponenten besorgen sich ihre Informationen selbst, anstatt mit Informationen versorgt zu werden. Die agierende Programmierung muß in Bezug auf den Datenaustausch umgekehrt wie die reagierende funktionale Programmierung ( $y=f(x_1, x_2, x_3, \dots)$ ) arbeiten ! Diese Grundlage ist zwingend, wenn ein Softwaresystem 'von außen' konfiguriert werden soll. Das existente Softwaresystem kann aus Unkenntnis über die zu adaptierenden Komponenten keine Daten uebergeben.





Ein Funktionsaufruf wird ueblich mit

*name funktion (Referenzierung auf zu uebergebende und empfangende Datencontainer).*

Beispiel: *ermittelgewicht (menge, mengenheit, artikelnr, gewicht)*

codiert. Die Funktion *ermittelgewicht* reagiert auf den Funktionsaufruf, agiert also nicht selbst. Basis der reagierenden funktionalen Programmierung ist das Versorgerprinzip, d.h. die angerufene Funktion wird von der aufrufenden Funktion mit Daten versorgt. Dieses Prinzip ist ebenfalls in objektorientierten Sprachen implementiert, auch wenn dem leistenden Objekt (Lieferantenobjekt) lediglich die Pointer auf das zu benutzende Dateninterface bereitgestellt werden. Die Pointeruebergabe ist eine Versorgungsidentifizierung fuer das zu benutzende Dateninterface und wird im „rufenden“ Objekt (=Kundenobjekt aus Sicht des leistenden Objektes) programmiert oder parametrisiert. Das Datenversorgerprinzip ist der gordische Knoten in Bezug auf die Konfiguration von Geschäftsprozessen mit untereinander kommunizierenden Komponenten.

Das Besorgerprinzip ist Voraussetzung für die OUTSIDE-INTEGRATION. Die oft realisierte Daten-, Ressourcen- und zeitaufwendige Integration über Pooldateien wird ersetzt durch die Datenbesorgung in der leistenden Komponente (say good by to files). Der Komponentenmonteur kann das Dateninterface (und damit Datenbesorgung und Datenentsorgung) in der leistenden Komponente ueberlagern.

Die DATENBE- und DATENENTSORGUNG von und in das Dateninterface der implementierungstechnisch rufenden Komponente (OBJECTline und dem Komponentenmonteur ist die rufende Komponente bekannt) kann durch zu spezifizierende Umleitungen erfolgen. Dies kann durch Komponentenhersteller fuer zu definierende Felder verboten werden. In diesem Fall schraenkt der Komponentenhersteller die beliebige Mehrfachverwendung seiner Komponente (und damit den Vermarktungserfolg) ein. Organismen funktionieren nach dem Prinzip der Vernunft. Ueber Monitorfunktionen sind Verletzungen dieses Prinzips nachweisbar. Die Realisierung der OUTSIDE-INTEGRATION erfordert, daß der herkoemlich uebliche Compiler im wesentlichen durch vom Metasystem zu interpretierende Repositorien ersetzt wird. Die Repositorien werden zum Injizierungszeitpunkt durch programmuebergreifende Analysen ermittelt und fuer das Runtime-System aufbereitet. Das Binärobjekt der Komponente wird 1. zum Zeitpunkt der Injizierung und 2. zum Ablaufzeitpunkt (wegen dynamisch aufzubauender Vererbungsgraphen) um Repositoryinformationen dynamisch ersetzt. Im Gegensatz zu einer ausschließlich schrittellenorientierten (und damit vom Komponentenher-

steller vorgedachten) Integration von Komponenten steht das Datensystem (die in Komponenten definierten Daten) der Aufgabenträger als Integrationstraeger fuer die Verknuepfung von Komponenten automatisch zur Verfügung. Damit sind verwendungsbezogene Schnittstellenerweiterungen der Komponente durch den Komponentenmonteur – unabhaengig von Komponentenlieferanten – injizierbar. Fuer ein Datenfeld wird die MESSAGE an den passenden Andockpunkt injiziert. Die MESSAGE kann auf eine vom Komponentenmonteur spezifizierte Adapterkomponente referenzieren (indirekte Injizierung), in der auf die leistende Komponente referenziert und ggf. Anweisungen fuer die Ueberlagerung von Dateninterfaces spezifiziert sind.

### Analyse der in einem Softwaresystem existenten Andockpunkte

Ein Softwaresystem wird auf passende verwendungsbezogene Slot's (Steckdosen) in den Aufgabenträgern des Softwaresystem's analysiert. Hierfuer wird zur Injizierungszeit ein Vererbungsparameter vorgegeben oder alternativ der Vererbungsparameter in der zu injizierenden Komponente eingetragen. Dieser Vererbungsparameter wird gegen alle existenten Andockpunkte von Aufgabenträgern des Softwaresystems verglichen. Aufgabenträger sind Bildschirmformate mit Eingabefeldern als Andockpunkte, Druckformate mit Ausgabefeldern als Andockpunkte, das Datensystem fuer persistent zu archivierende Daten mit Entity-Zugriffsoperationsarten als Andockpunkte. Die Interpretation der „Liste Adaptable Point's“ – die aus dem Softwaresystem abgerufen werden kann – ist wesentlich einfacher als die Analyse von Sourcen oder Handbuechern. Die Liste Adaptable Point's wird fuer die Automatische Injizierung von MESSAGE's in die ermittelten SLOT'S der Aufgabenträger verwendet.

## Interface-Syncronisation

Dynamisches Daten - Balancing zwischen den Dateninterfacen von zu verknuepfenden Komponenten. Hierbei wird zum Injizierungszeitpunkt (bei direkter Injizierung ohne Adapterkomponente) oder Runtimezeitpunkt (indirekte Injizierung ueber Adapterkomponente und Umleitung der Dateninterface) ueberpueft, ob Typenvertraeglichkeit vorliegt (syncronisierbare Feldtypen vorliegen).

Mit den genannten Eigenschaft koennen sich die zu adaptierende Komponenten 1. automatisch anpassen (Camaleon-Effekt der Interface) und 2. ein evtl. erforderlicher Adaptionsaufwand kann ausschliesslich in der leistenden Komponente (oder bei indirekter Injizierung in einer Adapterkomponente) implementiert werden.

Analog einer Datenbank ist damit eine komponentenbasierte Leistungsbank konstruierbar, aus der kundenspezifische Loesungen konfiguriert und z.B. von Beratern angepaest werden koennen. Diese Leistungsbank kann extern von Komponentenlieferanten erweitert werden ( Abb 3). Da die Adaption der Komponenten in existente Systeme von Komponentenmonteuren - und nicht vom Komponentenlieferanten oder Programmieren realisiert werden kann - wird die Vermarktung (z.B. via Internet) und Adaption unabhacngig von Aktivitaeten des Komponentenlieferanten ueber Komponenten-Distributoren erfolgen. Der hierraus resultierende Schneeballeffekt beinhaltet Chancen, die Komponentenlieferanten zur Bereitstellung sehr leistungsfuehiger Software motiviert. Dies ist der Anfang vom Ende monolithischer Systeme. Versorgerdienste werden als Black-Box-Dienste genutzt. Die erforderliche Qualitaetsicherung zu organisieren ist eher administrativer und nicht inonavitiver Natur. Software wird zu ganz normaler Handelsware.